A Cost Estimation Technique for Recursive Relational Algebra

Muideen Lawal, Pierre Genevès, Nabil Layaïda

To cite this version:

Muideen Lawal, Pierre Genevès, Nabil Layaïda. A Cost Estimation Technique for Recursive Relational Algebra. CIKM 2020 - 29th ACM International Conference on Information and Knowledge Management, Oct 2020, Virtual Event, France. pp.1-4, 10.1145/3340531.3417460. hal-03004218

HAL Id: hal-03004218
https://inria.hal.science/hal-03004218

Submitted on 13 Nov 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A Cost Estimation Technique for Recursive Relational Algebra

Muideen Lawal
Tyrex team
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble
France

Pierre Genevès
Tyrex team
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble
France

Nabil Layaïda
Tyrex team
Univ. Grenoble Alpes, CNRS, Inria, Grenoble INP, LIG, 38000 Grenoble
France

ABSTRACT

With the increasing popularity of data structures such as graphs, recursion is becoming a key ingredient of query languages in analytic systems. Recursive query evaluation involves an iterative application of a function or operation until some condition is satisfied. It is particularly useful for retrieving nodes reachable along deep paths in a graph. The optimization of recursive queries has remained a challenge for decades. Recently, extensions of Codd's classical relational algebra to support recursive terms and their optimisation gained renewed interest [10]. Query optimization crucially relies on enumeration of query evaluation plans and on cost estimation techniques. Cost estimation for recursive terms is far from trivial, and received less attention. In this paper, we propose a new cost estimation technique for recursive terms of the extended relational algebra. This technique allows to select an estimated cheapest query plan, in terms of computing resources usage *e.g. memory footprint, CPU and I/O* and evaluation time. We evaluate the effectiveness of our cost estimation technique on a set of recursive graph queries on both generated and real datasets of significant size, including Yago: a graph with more than 62 millions edges and 42 million nodes. Experiments show that our cost estimation technique improves the performance of recursive query evaluation on popular relational database engines such as PostgreSQL.

CCS CONCEPTS

• **Information systems → Database management system engines**; • **Theory of computation → Database theory**; **Database query processing and optimization (theory)**.

KEYWORDS

recursion; cost model; optimization; graph query

ACM Reference Format:
Muideen Lawal, Pierre Genevès, and Nabil Layaïda. 2020. A Cost Estimation Technique for Recursive Relational Algebra. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland*. ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3340531.3417460

This research has been partially supported by the ANR project CLEAR (ANR-16-CE25-0010).

Publication rights licensed to ACM. ACM acknowledges that this contribution was authored or co-authored by an employee, contractor or affiliate of a national government. As such, the Government retains a nonexclusive, royalty-free right to publish or reproduce this article, or to allow others to do so, for Government purposes only.
CIKM '20, October 19–23, 2020, Virtual Event, Ireland
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6859-9/20/10…$15.00
https://doi.org/10.1145/3340531.3417460

1 INTRODUCTION

In a typical query evaluation engine, a query is represented as a tree of operators denoting the query evaluation plan (*QEP*). The query engine includes a query optimizer, a crucial component in charge of searching for equivalent plans but in which operators are rearranged for efficiency purposes while preserving the semantics of the initial query. The query optimizer requires a cost estimation technique that selects a best plan, i.e. that provides a priori a better evaluation time and minimizes resources usage. For a given query sent to a query engine for evaluation, the optimizer first translates the query into a *QEP*, then generates a potentially huge number of equivalent *QEPs*. These plans have different query evaluation times and resources utilization depending on the cardinality, selectivity and the order in which operators are arranged in the tree. Depending on the plan selected by the optimizer for execution, the disparity in query time might range from a few milliseconds to minutes and even days. The essence of a cost model in query processing is to estimate and compare the cost of executing a query using different query plans and choose the plan with the "cheapest" cost [11]. Choosing a query plan with minimum cost for computation minimizes the total resources required [13]. Recursive queries expresses a category of complex queries that involve iterative application of a function or operation until some condition is satisfied – known as the *fixpoint*. A variety of studies has been conducted on this class of queries including [5, 9, 11] and more recently [7, 10, 14]. One of the most difficult tasks in estimating the cost of a recursive query is determining the number of iterative steps needed for the iteration to converge. Many cost models for recursive queries usually assume a constant number of iterative steps (the constant 10 for instance as found in [3, 7]). This can often lead to poor cost estimation for query execution plans.

Our main contribution is to propose a new technique for estimating *(i)* the number of iterative steps and *(ii)* the computation cost for recursive query evaluation. We experimentally demonstrate the benefits of this technique for recursive queries evaluated with PostgreSQL.

2 COST ESTIMATION

We propose a cost estimation technique suitable for recursive terms of the extended relational algebra [10] following the initial idea first described in the seminal approach of *System R* [13] followed by extensive works on the topic [6, 8, 12, 17].

The input of the cost estimation function is a term φ of the recursive relational algebra proposed in [10], which corresponds to Codd's classical relational algebra extended with a fixpoint operator, and whose syntax is:

$$
\begin{array}{llr}
\varphi & ::= & \text{term} \\
 & R & \text{relation variable} \\
| & |c \to v| & \text{constant} \\
| & \varphi_1 \cup \varphi_2 & \text{union} \\
| & \varphi_1 \bowtie \varphi_2 & \text{join} \\
| & \varphi_1 \triangleright \varphi_2 & \text{antijoin} \\
| & \sigma_{\mathfrak{f}}(\varphi) & \text{filtering} \\
| & \rho_a^b(\varphi) & \text{renaming} \\
| & \widetilde{\pi}_a(\varphi) & \text{anti-projection} \\
| & \mu X. \underbrace{\varphi}_{\text{constant part}} \cup \underbrace{(R \bowtie X)}_{\text{recursive part}} & \text{fixpoint}
\end{array}
$$

Recursive terms are expressed using the fixpoint operator, that contains two parts: the *constant part* and the *recursive part*. The *recursive part* is executed several times until it no longer retrieves further results. The *constant part*, executed just once, provides the initial results used as a starting point for the recursion (see [10] for a formal semantics).

Based on the aforementioned syntax, we define a cost estimation function for a term φ as:

$$cost(\varphi) = \big(evalCost, rowCount\big) \tag{1}$$

where *evalCost* is the estimated computation cost and *rowCount* is the estimated size of the result, i.e. the number of tuples returned. The function $cost(\varphi)$ is defined recursively using a bottom up approach, starting from the tree leaves (constants and relational variables).

2.1 Standard non-recursive constructs

For all cases except the fixpoint operator, we reuse standard cost estimation techniques known from state-of-the-art works on the topic [6, 8, 12, 13, 17].

Cardinality estimation is essential for making accurate cost estimation [6, 12]. We consider a set of statistics initially computed from relation variables, which includes the number of tuples (rows) in the relation and the number of distinct values per column (attribute). Changes in cardinality are then tracked and propagated within the query tree. Selectivity refers to the set of tuples in a relation that satisfy an applicable predicate [8, 13]. The *Selectivity* estimation mostly relies on the number of distinct values per attribute of the relation. We adopt the work of [17] for calculating the *number of distinct value* of e.g. joined columns which is later reused as selectivity factor for the join operator.

2.2 Fixpoint operator

One challenging part consists in determining when the recursion terminates. This is useful for estimating the number of rows returned and subsequently the cost of the whole fixpoint operator.

To estimate the cost of a *fixpoint*, we follow the steps below:

(1) we start from X which is initially an empty relation ($\emptyset$), we substitute X into the equation of the fixpoint and perform a union, the whole fixpoint term then reduces to $\varphi \cup (R \bowtie \emptyset) = \varphi$, thus we have $rowCount = rowCount(\varphi)$
(2) at this step, the value of X is now $\varphi \cup (R \bowtie \varphi)$. Given the cardinality of φ and R and the selectivity factor of the join, by substituting the result of X (i.e $R \bowtie \varphi$), we compute the *evalcost* and *rowcount* of this step;
(3) by iterative substitution of X, the computation continues repeatedly until some step N such that the result size is less or equal than the initial selectivity factor, i.e. $rowCount \leq Sel$. At this point, we estimate that the maximum number of iterations has been reached and that the iteration terminates.

This estimation relies on several assumptions, that are inspired by the so-called *semi-naïve* evaluation of transitive closures found in the literature [1, 5, 7, 9]. In particular, we assume that only the new results generated by an iteration are used for the next iteration and that the number of tuples reduces until a maximum number of iterations $\mathbb{N}$ is reached. At each step, the result size is reduced by a factor s which we compute from the base case of the *fixpoint (i.e. $R \bowtie X$).* We estimate the number $\mathbb{N}$ of iterations as:

$$\mathbb{N} = \log_s(K) \tag{2}$$

where $K = rowCount(R \bowtie X)$ is the estimated number of tuples in the recursive part of the fixpoint.

We now know the number $\mathbb{N}$ of iterative steps. We now proceed to compute the cost of the overall fixpoint term. Let $c_1 = cost(\varphi)$, $c_2 = cost(R)$ respectively:

- $c_1.evalCost$ = cost of computing φ
- $c_1.rowCount$ = number of tuples in φ
- $c_2.evalCost$ = cost of computing the recursive relation R
- $c_2.rowCount$ = number of tuples in the recursive relation R

The evaluation cost for a *fixpoint* is given as;

$$evalCost \ = \ c_1.evalCost + (c_2.evalCost \times \mathbb{N}) + rowCount \tag{3}$$

At *step 1* above, X is empty, then the *rowCount* at step 1 is;

$$rowCount(X) = c_1.rowCount \tag{4}$$

The evaluation cost at this point is estimated as $c_1.evalCost$ and $c_2.rowCount$ respectively. We then estimate the join size *i.e.* $rowCount(R \bowtie \varphi)$

$$rowCount(R \bowtie \varphi) = c_2.rowCount \times c_1.rowCount \times \mathbb{S}el \tag{5}$$

The evaluation cost is estimated as:

$$
\begin{aligned}
evalCost = {} & \text{cost of computing const. part} \\
& + (\mathbb{N} \times \text{cost of scanning rec. part}) \\
& + \text{cost of gathering the results}
\end{aligned} \tag{6}
$$

We estimate the *cost of gathering the results* $Cost_{res}$ as the maximum of the cardinality of relation E (i.e. $c_1.rowCount$) and $rowCount(R \bowtie E)$ in order to avoid misestimation.

$$Cost_{res} = max(c_1.rowCount, rowCount(R \bowtie E)) \tag{7}$$

The result size (*rowCount*) of a *fixpoint* operator is estimated as;

$$rowCount = c_1.rowCount + rowCount(R \bowtie E)) \tag{8}$$

3 EXPERIMENTAL EVALUATION

3.1 Experimental Setup

We conduct several experiments to assess the effectiveness of our cost estimation technique by evaluating recursive graph queries (that are union of conjunctive regular path queries [4]) using their translation (found in [10]) into the recursive relational algebra. A graph query is first translated into a term φ, then all equivalent terms are exhaustively enumerated resulting in a plan space $\mathcal{P}$.

Finally among all terms in $\mathcal{P}$, the term φ which minimizes a cost estimation function f is retained, and executed.

We evaluate queries using two systems corresponding to two settings:

- *System P* is the popular PostgreSQL system [15], where f is the function that returns the cost estimated by PostgreSQL using the explain API;
- *System P'* is also the PostgreSQL system, but where f is the cost function that we propose in this paper.

The comparison between the two settings is fair since the only difference is the cost estimation function.

We experiment with two kinds of datasets: a real-world dataset: Yago2s [16] and two synthetic datasets (Shop and Uniprot) generated by the *GMark* system [2]: We report the results that we have

		Column cardinality		
Dataset	cardinality	src	trg	label
Yago2s	62,643,951	35,165,791	8,572,450	83
Shop	93,398	24,842	46,038	80
Uniprot	385,447	43000	61,927	7

* column cardinality is the number of distinct values in each column

obtained for the 20 queries over Yago2s found in [10], and for 10 randomly generated queries for each synthetic dataset.

Experiments on the Yago dataset were conducted on a 128 GB RAM server with *2 Intel Xeon E5-2630 v4 CPUs (2.20 GHz, 20 cores each)*. All other experiments reported were conducted on a 16GB RAM *Intel(R) Core(TM) i7-4980HQ CPU @ 2.80GHz* machine.

3.2 Results: relative query performance

Fig. 1 shows the times spent by the two systems for evaluating queries on the synthetic datasets. Results show that both systems evaluate all the queries in a comparable amount of time: all queries are evaluated by both systems in less than 0.3 seconds. Specifically, *System P'* outperforms *System P* for 19 out of the 20 queries. For the remaining case of q_4 on the Shop dataset, *System P* performs better by 150 milliseconds.

Fig. 2 shows the times spent for evaluating the 20 queries of [10] on the real-world Yago2s dataset [16]. Results show that *System P'* outperforms *System P*. In particular, System P could not answer queries $q_8, q_{13}, q_{17}, q_{20}$ within the allowed time frame of 15 minutes, while System P' evaluates these queries in 146, 108, 34 and 14 seconds, respectively. For the 16 other queries remaining, the results of System P' are comparable or even slightly better than System P'. This illustrates the practical interest of the refined cost estimation.

3.3 Results: ranking of cost estimations

We also run all equivalent terms of the plan space $\mathcal{P}$ that are generated by the optimizer, in order to assess how our term-picking function compares to the best terms of $\mathcal{P}$: the ones with the minimum actual query times.

Fig. 3a and Fig. 3b show the number of queries for which the plan picked by each system was the one with minimum cost, or in the 15th percentile, the 25th percentile, etc. among all plans in $\mathcal{P}$ ranked in increasing order of actual running times. We observe that System P' picks more efficient terms more often.

4 CONCLUSION AND FUTURE WORKS

We propose a refined cost estimation technique for recursive terms in the relational algebra. Experiments with a prototype of the approach show that this improves the performance of recursive query evaluation on popular relational database engines such as PostgreSQL. This contribution can be implemented in mainstream database management systems supporting recursive query evaluation. It can be helpful to improve the support of query evaluation for graph structures that are becoming ubiquitous.

REFERENCES

[1] Foto N Afrati, Vinayak Borkar, Michael Carey, Neoklis Polyzotis, and Jeffrey D Ullman. 2011. Map-reduce extensions and recursive queries. In Proceedings of the 14th international conference on extending database technology. 1–8.

[2] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat. 2017. gMark: Schema-Driven Generation of Graphs and Queries. IEEE Transactions on Knowledge and Data Engineering 29, 4 (2017), 856–869.

[3] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J Mior, and Daniel Lemire. 2018. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In Proceedings of the 2018 International Conference on Management of Data. 221–230.

[4] Angela Bonifati, George Fletcher, Hannes Voigt, and Nikolay Yakovets. 2018. Querying graphs. Synthesis Lectures on Data Management 10, 3 (2018), 1–184.

[5] Filippo Cacace, Stefano Ceri, and Maurice AW Houtsma. 1990. An overview of parallel strategies for transitive closure on algebraic machines. In Workshop on Parallel Database Systems. Springer, 44–62.

[6] Surajit Chaudhuri. 1998. An overview of query optimization in relational systems. In Proceedings of the seventeenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems. ACM, 34–43.

[7] Stephan Ewen, Kostas Tzoumas, Moritz Kaufmann, and Volker Markl. 2012. Spinning fast iterative data flows. arXiv preprint arXiv:1208.0088 (2012).

[8] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Arun N Swami. 1996. Selectivity and cost estimation for joins based on random sampling. J. Comput. System Sci. 52, 3 (1996), 550–569.

[9] Yannis E Ioannidis. [n.d.]. On the computation of the transitive closure of relational operators.

[10] Louis Jachiet, Pierre Genevès, Nils Gesbert, and Nabil Layaïda. 2020. On the Optimization of Recursive Relational Queries: Application to Graph Queries. In Proceedings of the ACM SIGMOD International Conference on Management of Data. https://hal.inria.fr/hal-01673025v5/document

[11] Rosana SG Lanzelotte, Patrick Valduriez, and Mohamed Zaït. 1992. Optimization of object-oriented recursive queries using cost-controlled strategies. In ACM SIGMOD Record, Vol. 21. ACM, 256–265.

[12] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really? Proceedings of the VLDB Endowment 9, 3 (2015), 204–215.

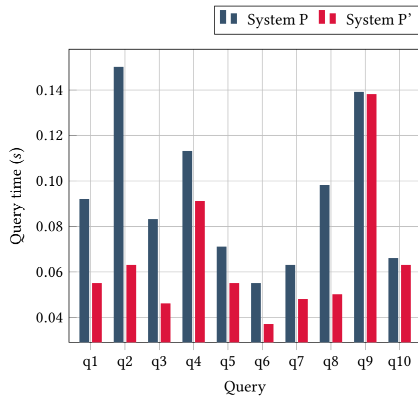
[13] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access path selection in a relational database management system. In Proceedings of the 1979 ACM SIGMOD international conference on Management of data. ACM, 23–34.

[14] Alexander Shkapsky, Mohan Yang, Matteo Interlandi, Hsuan Chiu, Tyson Condie, and Carlo Zaniolo. 2016. Big data analytics with datalog queries on spark. In Proceedings of the 2016 International Conference on Management of Data. ACM, 1135–1149.

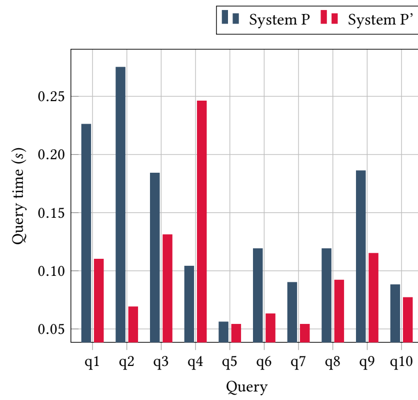
[15] Michael Stonebraker and Lawrence A Rowe. 1986. The design of POSTGRES. ACM Sigmod Record 15, 2 (1986), 340–355.

[16] Fabian M. Suchanek, Gjergji Kasneci, and Gerhard Weikum. 2007. Yago: A Core of Semantic Knowledge. In 16th International Conference on the World Wide Web. 697–706.

[17] Arun Swami and K Bernhard Schiefer. 1994. On the estimation of join result sizes. In International Conference on Extending Database Technology. Springer, 287–300.



(a) GMark [2] Uniprot



(b) GMark [2] Shop

Figure 1: Query evaluation times for queries on test datasets

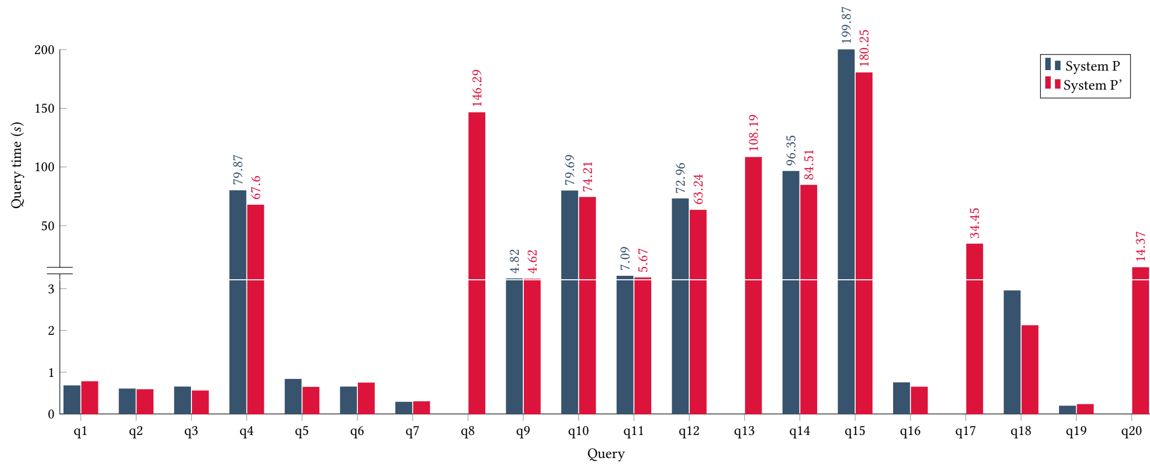
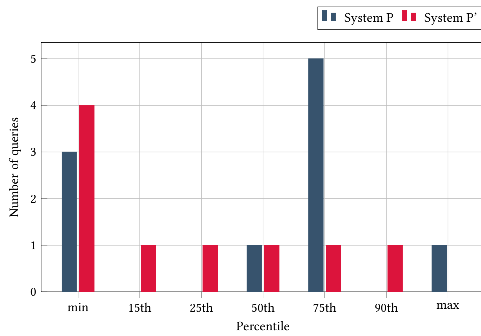
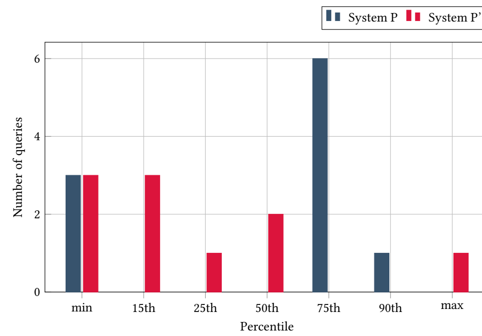


Figure 2: Query evaluation time for queries on Yago [16]



(a) Uniprot [2]



(b) Shop [2]

Figure 3: Query rank by percentile for GMark[2]